python-archinstall

Release v2.1.0

Sep 07, 2021

Running the installer

python-archinstall *(or, archinstall for short)* is a helper library to install Arch Linux and manage services, packages and other things.
It comes packaged with different pre-configured installers, such as the **'Guided installation'_** installer.

A demo can be viewed here: https://www.youtube.com/watch?v=9Xt7X_Iqg6E which uses the default guided installer.

Some of the features of Archinstall are:

- **No external dependencies or installation requirements.** Runs without any external requirements or installation processes.
- **Single threaded and context friendly.** The library always executed calls in sequential order to ensure installation-steps don't overlap or executes in the wrong order. It also supports *(and uses)* context wrappers to ensure things such as *sync* are called so data and configurations aren't lost.
- **Supports standalone executable** The library can be compiled into a single executable and run on any system with or without Python. This is ideal for live mediums that don't want to ship Python as a big dependency.

CHAPTER 1

Guided installation

This is the default scripted installation you'll encounter on the official Arch Linux Archinstall package as well as the unofficial ISO found on https://archlinux.life. It will guide your through a very basic installation of Arch Linux.

The installer has two pre-requisites:

- A Physical or Virtual machine to install on
- An active internet connection prior to running archinstall

Warning: A basic understanding of how machines, ISO-files and command lines are needed. Please read the official Arch Linux Wiki *('https://wiki.archlinux.org/ <https://wiki.archlinux.org/>'_ to learn more)*

Note: There are some limitations with the installer, such as that it will not configure WiFi during the installation procedure. And it will not perform a post-installation network configuration either. So you need to read up on Arch Linux networking to get that to work.

1.1 Running the guided installation

Note: Due to the package being quite new, it might be required to update the local package list before installing or continuing. Partial upgrades can cause issues, but the lack of dependencies should make this perfectly safe:

To install archinstall and subsequently the guided installer, simply do the following:

```
pacman -S python-archinstall
```

And to run it, execute archinstall as a Python module:

```
python -m archinstall --script guided
```

The `--script guided` argument is optional as it's the default behavior.
But this will start the process of guiding you through a installation of a quite minimal Arch Linux experience.

1.2 Installing directly from a configuration file

Note: Edit the following json according to your needs, save this as a json file, and provide the local or remote path (URL)

```
{
    "audio": "pipewire",
    "bootloader": "systemd-bootctl",
    "custom-commands": [
        "cd /home/devel; git clone https://aur.archlinux.org/paru.git",
        "chown -R devel:devel /home/devel/paru",
        "usermod -aG docker devel"
    ],
    "!encryption-password": "supersecret",
    "filesystem": "btrfs",
    "gfx_driver": "VMware / VirtualBox (open-source)",
    "harddrive": {
        "path": "/dev/nvme0n1"
    },
    "hostname": "development-box",
    "kernels": [
        "linux"
    ],
    "keyboard-language": "us",
    "mirror-region": "Worldwide",
    "nic": {
        "NetworkManager": true
    },
    "ntp": true,
    "packages": ["docker", "git", "wget", "zsh"],
    "profile": "gnome",
    "services": ["docker"],
    "superusers": {
        "devel": {
            "!password": "devel"
        }
    },
    "sys-encoding": "utf-8",
    "sys-language": "en_US",
    "timezone": "US/Eastern",
    "users": {}
}
```

To run it, execute archinstall as a Python module:

```
python -m archinstall --config <local path or remote URL>
```

Key	Values/Description	Description	Required
audio	pipewire/pulseaudio	Audioserver to be installed	No
bootloader	systemd-bootctl/grub-install	Bootloader to be installed	Yes
custom-commands	[<command1>, <command2>, . . .]	Custom commands to be run post install	No
!encryption-password	any	Password to encrypt disk, not encrypted if password not provided	No
filesystem	ext4 / btrfs / fat32 etc.	Filesystem for root and home partitions	Yes
gfx_driver	“VMware / VirtualBox (open-source)” or “Nvidia” or “Intel (open-source)” or “AMD / ATI (open-source)” or “All open-source (default)”	Graphics Drivers to install	No
harddrive	{ “path”: <path of device> }	Path of device to be used	Yes
hostname	any	Hostname of machine after installation	Yes
kernels	[“kernel1”, “kernel2”]	List of kernels to install eg: linux, linux-lts, linux-zen etc	Atleast 1
keyboard-language	2 letter code for your keyboard language	eg: us, de etc	Yes
mirror-region	{“<Region Name>”: { “Mirror Name”: True/False}, ..}	List of regions and mirrors to use	Yes
nic	{ NetworkManager: <boolean>, nic: <nic name> }		Yes
ntp	<boolean>	Set to true to set-up ntp post install	No
packages	[“package1”, “package2”, ..]	List of packages to install post-installation	No
profile	Name of the profile to install	Profiles are present in profiles/, use the name of a profile to install it	No
!root-password	any	The root account password	No
services	[“service1”, “service2”, ..]	Services to enable post-installation	No
sys-encoding	“utf-8”	Set to change system encoding post-install, ignored if –advanced flag is not passed	No
sys-language	“en_US”	Set to change system language post-install, ignored if –advanced flag is not passed	No
superusers	{ “<username>”: { “!password”: “<password>”}, ..}	List of superuser credentials, see configuration for reference	Yes, if root account password is not provided
timezone	Timezone to configure in installation	Timezone eg: UTC, Asia/Kolkata etc.	Yes
users	{ “<username>”: { “!password”: “<password>”}, ..}	List of regular user credentials, see configuration for reference	Yes, can be {}

CHAPTER 2

Description individual steps

Below is a description of each individual steps in order.

2.1 keyboard languages

Default is `us`.

A short list of the most common layouts are presented.
Entering `?` and pressing enter enables a search mode where additional keyboard layouts can be found.

In search mode, you'll find things like:

- `sv-latin1` for swedish layouts

2.2 Mirror region selection

Default is `auto detect best mirror`

Leaving this blank should enable the most appropriate mirror for you.
But if you want to override and use only one selected region, you can enter one in this step.

As an example:

- `Sweden` *(with a capital :code:'S')* will only use mirrors from Sweden.

2.3 Selection of drive

There is no default for this step and it's a required step.

Warning:

The selected device will be wiped completely!

Make sure you select a drive that will be used only by Arch Linux.
(Future versions of archinstall will support multiboot on the same drive and more complex partition setups)

Select the appropriate drive by selecting it by number or full path.

2.4 Disk encryption

Selecting a disk encryption password enables disk encryption for the OS partition.

Note: This step is highly recommended for most users, skipping this step comes with some risk and you are obligated to read up on why you would want to skip encryption before deciding to opt-out.

Warning: This step does require at least 1GB of free RAM during boot in order to boot at all. Keep this in mind when creating virtual machines. It also only encrypts the OS partition - not the boot partition *(it's not full disk encryption)*.

2.5 Hostname

Default is `Archinstall`

The hostname in which the machine will identify itself on the local network. This step is optional, but a default hostname of *Archinstall* will be set if none is selected.

2.6 Root password

Warning:

Setting a root password disables sudo permissions for additional users.
It's there for **recommended to skip this step**!

This gives you the option to re-enable the `root` account on the machine. By default, the `root` account on Arch Linux is disabled and does not contain a password.

You are instead recommended to skip to the next step without any input.

2.7 Super User (sudo)

Warning: This step only applies if you correctly skipped *the previous step* which also makes this step mandatory.

If the previous step was skipped, and only if it is skipped. This step enables you to create a `sudo` enabled user with a password.

Note: The sudo permission grants `root`-like privileges to the account but is less prone to for instance guessing admin account attacks. You are also less likely to mess up system critical things by operating in normal user-mode and calling *sudo* to gain temporary administrative privileges.

2.8 Pre-programmed profiles

You can optionally choose to install a pre-programmed profile. These profiles might make it easier for new users or beginners to achieve a traditional desktop environment as an example.

There is a list of profiles to choose from. If you are unsure of what any of these are, research the names that show up to understand what they are before you choose one.

Note:

Some profiles might have sub-dependencies that will ask you to select additional profiles.
For instance the `desktop` profile will create a secondary menu to select a graphical driver. That graphical driver might have additional dependencies if there are multiple driver vendors.

Simply follow the instructions on the screen to navigate through them.

2.9 Additional packages

Some additional packages can be installed if need be. This step allows you to list *(space separated)* officially supported packages from the package database at https://archlinux.org/packages/.

2.10 Network configuration

In this step is optional and allows for some basic configuration of your network.
There are two main options and two sub-options, the two main ones are:

- Copy existing network configuration from the ISO you're working on
- Select **one** network interface to configure

If copying existing configuration is chosen, no further configuration is needed.

The installer will copy any wireless *(based on :code:'iwd')* configurations and `systemd-networkd` configuration set up by the user or the default system configuration.

If a interface was selected instead, a secondary option will be presented, allowing you to choose between two options:

- Automatic DHCP configuration of IP, DNS and Gateway
- Static IP configuration that further will ask some mandatory questions

2.11 Configuration verification

Before the installer continues, and this is only valid for the **guided installation**.
The chosen configuration will be printed on the screen and you have the option to verify it.

After which you can press `Enter` can be pressed in order to start the formatting and installation process.

> **Warning:** After a 5 second countdown, the selected drive will be permanently erased and all data will be lost.

2.12 Post installation

Once the installation is complete, green text should appear saying that it's safe to *reboot*, which is also the command you use to reboot.

CHAPTER 3

Discord

There's a discord channel which is frequent by some contributors.

To join the server, head over to https://discord.gg/cqXU88y's server and join in. There's not many rules other than common sense and treat others with respect.

There's the *@Party Animals* role if you want notifications of new releases which is posted in the *#Release Party* channel. Another thing is the *@Contributors* role which you can get by writing *!verify* and verify that you're a contributor.

Hop in, I hope to see you there! :)

CHAPTER 4

Issue tracker & bugs

Issues and bugs should be reported over at https://github.com/archlinux/archinstall/issues.

General questions, enhancements and security issues can be reported over there too. For quick issues or if you need help, head over the to the Discord server which has a help channel.

CHAPTER 5

Submitting a help ticket

When submitting a help ticket, please include the `/var/log/archinstall/install.log`.
It can be found both on the live ISO but also in the installed filesystem if the base packages was strapped in.

There are additional worker files, these worker files contain individual command input and output.
These worker files are located in `~/.cache/archinstall/` and does not need to be submitted by default when submitting issues.

Warning: Worker log-files *may* contain sensitive information such as **passwords** and **private information**. Never submit these logs without going through them manually making sure they're good for submission. Or submit parts of it that's relevant to the issue itself.

CHAPTER 6

Python library

Archinstall ships on PyPi as archinstall. But the library can be installed manually as well.

Warning: This is not required if you're running archinstall on a pre-built ISO. The installation is only required if you're creating your own scripted installations.

6.1 Using pacman

Archinstall is on the official repositories.

To install both the library and the archinstall script:

```
sudo pacman -S archinstall
```

Or, to install just the library:

```
sudo pacman -S python-archinstall
```

6.2 Using PyPi

The basic concept of PyPi applies using *pip*. Either as a global library:

```
sudo pip install archinstall
```

Or as a user module:

```
pip --user install archinstall
```

Which will allow you to start using the library.

6.3 Manual installation

You can either download the github repo as a zip archive. Or you can clone it, we'll clone it here but both methods work the same.

```
git clone https://github.com/archlinux/archinstall
```

Either you can move the folder into your project and simply do

```
import archinstall
```

Or you can use setuptools to install it into the module path.

```
sudo python setup.py install
```

CHAPTER 7

Binary executable

Archinstall can be compiled into a standalone executable. For Arch Linux based systems, there's a package for this called archinstall.

Warning: This is not required if you're running archinstall on a pre-built ISO. The installation is only required if you're creating your own scripted installations.

7.1 Using pacman

Archinstall is on the official repositories.

```
sudo pacman -S archinstall
```

7.2 Using PKGBUILD

The source contains a binary PKGBUILD which can be either copied straight off the website. Or cloned using *git clone https://github.com/Torxed/archinstall*.

Once you've obtained the *PKGBUILD*, building it is pretty straight forward.

```
makepkg -s
```

Which should produce a *archinstall-X.x.z-1.pkg.tar.zst* that can be installed using:

```
sudo pacman -U archinstall-X.x.z-1.pkg.tar.zst
```

Note: For a complete guide on the build process, please consult the PKGBUILD on ArchWiki.

7.3 Manual compilation

You can compile the source manually without using a custom mirror or the *PKGBUILD* that is shipped. Simply clone or download the source, and while standing in the cloned folder *./archinstall*, execute:

```
nuitka3  --standalone --show-progress archinstall
```

This requires the nuitka package as well as *python3* to be installed locally.

CHAPTER 8

Python module

Archinstall supports running in module mode. The way the library is invoked in module mode is limited to executing scripts under the **example** folder.

It's there for important to place any script or profile you wish to invoke in the examples folder prior to building and installing.

8.1 Pre-requisites

We'll assume you've followed the *Manual installation* method. Before actually installing the library, you will need to place your custom installer-scripts under *./archinstall/examples/* as a python file.

More on how you create these in the next section.

> **Warning:** This is subject to change in the future as this method is currently a bit stiff. The script path will become a parameter. But for now, this is by design.

8.2 Creating a script

Lets create a *test_installer* - installer as an example. This is assuming that the folder *./archinstall* is a git-clone of the main repo. We begin by creating *./archinstall/examples/test_installer.py*. The placement here is important later.

This script can now already be called using *python -m archinstall test_installer* after a successful installation of the library itself. But the script won't do much. So we'll do something simple like list all the hard drives as an example.

To do this, we'll begin by importing *archinstall* in our *./archinstall/examples/test_installer.py* and call some functions.

```
import archinstall

all_drives = archinstall.list_drives()
print(all_drives)
```

This should print out a list of drives and some meta-information about them. As an example, this will do just fine.

Now, go ahead and install the library either as a user-module or system-wide.

8.3 Calling a module

Assuming you've followed the example in *Creating a script*, you can now safely call it with:

```
python -m archinstall test_installer
```

This should now print all available drives on your system.

Note: This should work on any system, not just Arch Linux based ones. But note that other functions in the library relies heavily on Arch Linux based commands to execute the installation steps. Such as *arch-chroot*.

CHAPTER 9

Binary executable

Warning: The binary option is limited and stiff. It's hard to modify or create your own installer-scripts this way unless you compile the source manually. If your usecase needs custom scripts, either use the pypi setup method or you'll need to adjust the PKGBUILD prior to building the arch package.

The binary executable is a standalone compiled version of the library. It's compiled using nuitka with the flag *–standalone*.

9.1 Executing the binary

As an example we'll use the guided installer. To run the *guided* installed, all you have to do *(after installing or compiling the binary)*, is run:

```
./archinstall guided
```

As mentioned, the binary is a bit rudimentary and only supports executing whatever is found directly under *./archinstall/examples*. Anything else won't be found. This is subject to change in the future to make it a bit more flexible.

CHAPTER 10

archinstall.Installer

The installer is the main class for accessing a installation-instance. You can look at this class as the installation you have or will perform.

Anything related to **inside** the installation, will be found in this class.

`archinstall.`**`Installer`**(*target*, *, *base_packages=None*, *kernels=None*)

Installer() is the wrapper for most basic installation steps. It also wraps `pacstrap()` among other things.

Parameters

- **`partition`** (class:*archinstall.Partition*) – Requires a partition as the first argument, this is so that the installer can mount to *mountpoint* and strap packages there.
- **`boot_partition`** (class:*archinstall.Partition*) – There's two reasons for needing a boot partition argument, The first being so that *mkinitcpio* can place the *vmlinuz* kernel at the right place during the *pacstrap* or *linux* and the base packages for a minimal installation. The second being when `add_bootloader()` is called, A *boot_partition* must be known to the installer before this is called.
- **`profile`** (*`str, optional`*) – A profile to install, this is optional and can be called later manually. This just simplifies the process by not having to call `install_profile()` later on.
- **`hostname`** (*`str, optional`*) – The given /etc/hostname for the machine.

CHAPTER 11

archinstall.Profile

This class enables access to pre-programmed profiles. This is not to be confused with *archinstall.Application* which is for pre-programmed application profiles.

Profiles in general is a set or group of installation steps. Where as applications are a specific set of instructions for a very specific application.

An example would be the *(currently fictional)* profile called *database*. The profile *database* might contain the application profile *postgresql*. And that's the difference between *archinstall.Profile* and *archinstall.Application*.

archinstall.**Profile**(*installer*, *path*, *args=None*)

CHAPTER 12

archinstall.Application

This class enables access to pre-programmed application configurations. This is not to be confused with *archinstall.Profile* which is for pre-programmed profiles for a wider set of installation sets.

archinstall.**Application**(*installer*, *path*, *args=None*)

> **Warning:** All these helper functions are mostly, if not all, related to outside-installation-instructions. Meaning the calls will affect your current running system - and not touch your installed system.

CHAPTER 13

Profile related helpers

archinstall.**list_profiles**(*filter_irrelevant_macs=True*, *subpath=''*, *filter_top_level_profiles=False*)

CHAPTER 14

Packages

archinstall.**find_package**(*name*)

Finds a specific package via the package database. It makes a simple web-request, which might be a bit slow.

Be .. autofunction:: archinstall.find_packages

CHAPTER 15

Locale related

archinstall.**list_keyboard_languages**()

archinstall.**search_keyboard_layout**(*layout*)

archinstall.**set_keyboard_language**(*locale*)

CHAPTER 16

Services

archinstall.**service_state**(*service_name: str*)

CHAPTER 17

Mirrors

archinstall.**filter_mirrors_by_region**(*regions*, *destination='/etc/pacman.d/mirrorlist'*, **args*, ***kwargs*)

This function will change the active mirrors on the live medium by filtering which regions are active based on *regions*.

Parameters **regions** (*str*) – A series of country codes separated by ,. For instance *SE,US* for sweden and United States.

archinstall.**add_custom_mirrors**(*mirrors: list*, **args*, ***kwargs*)

This will append custom mirror definitions in pacman.conf

Parameters **mirrors** (*dict*) – A list of mirror data according to: *{'url': 'http://url.com', 'signcheck': 'Optional', 'signoptions': 'TrustAll', 'name': 'testmirror'}*

archinstall.**insert_mirrors**(*mirrors*, **args*, ***kwargs*)

This function will insert a given mirror-list at the top of */etc/pacman.d/mirrorlist*. It will not flush any other mirrors, just insert new ones.

Parameters **mirrors** (*dict*) – A dictionary of *{'url' : 'country', 'url2' : 'country'}*

archinstall.**use_mirrors**(*regions: dict*, *destination='/etc/pacman.d/mirrorlist'*)

archinstall.**re_rank_mirrors**(*top=10*, **positionals*, ***kwargs*)

archinstall.**list_mirrors**()

CHAPTER 18

Disk related

archinstall.**BlockDevice**(*path*, *info=None*)

archinstall.**Partition**(*path: str*, *block_device: archinstall.lib.disk.BlockDevice*, *part_id=None*, *size=-1*, *filesystem=None*, *mountpoint=None*, *encrypted=False*, *autodetect_filesystem=True*)

archinstall.**Filesystem**(*blockdevice*, *mode*)

archinstall.**device_state**(*name*, **args*, ***kwargs*)

archinstall.**all_disks**(**args*, ***kwargs*)

CHAPTER 19

Luks (Disk encryption)

archinstall.**luks2**(*partition*, *mountpoint*, *password*, *key_file=None*, *auto_unmount=False*, **args*, ***kwargs*)

CHAPTER 20

Networking

archinstall.**list_interfaces**(*skip_loopback=True*)

CHAPTER 21

General

archinstall.**log**(**args*, ***kwargs*)

archinstall.**locate_binary**(*name*)

CHAPTER 22

Exceptions

archinstall.**RequirementError**()
Common base class for all exceptions

archinstall.**DiskError**()
Common base class for all exceptions

archinstall.**ProfileError**()
Common base class for all exceptions

archinstall.**SysCallError**(*message*, *exit_code*)
Common base class for all exceptions

archinstall.**ProfileNotFound**()
Common base class for all exceptions

archinstall.**HardwareIncompatibilityError**()
Common base class for all exceptions

archinstall.**UserError**()
Common base class for all exceptions

archinstall.**ServiceException**()
Common base class for all exceptions

Index